Applied Visions, Inc.

SBIR AF97-043

Phase II

Network Security Visualization

Final Technical Report

27 September 1999

Applied Visions, Inc.
6 Bayview Avenue
Northport, NY 11768

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (FROM - TO)
27-09-1999	SBIR Report	xx-xx-1999 to xx-xx-1999

4. TITLE AND SUBTITLE
Network Security Visualization Final Technical Report, Phase II
Unclassified

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME AND ADDRESS
Applied Visions, Inc.
6 Bayview Avenue
Northport, NY11768

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME AND ADDRESS
,

10. SPONSOR/MONITOR'S ACRONYM(S)
11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APUBLIC RELEASE
,

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The application of interactive, three-dimensional viewing techniques to the representation of security-related, computer network status and events is expected to improve the timeliness and efficiency of monitoring network security. This document outlines the final status and content of a functional prototype system developed under this Phase II SBIR contract.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19. NAME OF RESPONSIBLE PERSON
			Public Release	35	Fenster, Lynn lfenster@dtic.mil
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER International Area Code Area Code Telephone Number 703767-9007 DSN 427-9007

Table of Contents

1 Scope

The application of interactive, three-dimensional viewing techniques to the representation of security-related, computer network status and events is expected to improve the timeliness and efficiency of monitoring network security. This document outlines the final status and content of a functional prototype system developed under this Phase II SBIR contract.

1.1 Applicable Documents

The following technical reports have been delivered as part of this contract and serve as detailed references for the contents of this report.

“SBIR AF97-043 Network Security Visualization - Phase II User Manual”, Applied Visions, Inc., 1999.

“SBIR AF97-043 Network Security Visualization - Phase II Installation Instructions”, Applied Visions, Inc., 1999.

“SBIR AF97-043 Network Security Visualization - Phase II Software Design Specification”, Applied Visions, Inc., 1999.

“SBIR AF97-043 Network Security Visualization - Phase II User Interface Specification”, Applied Visions, Inc., 1999.

“SBIR AF97-043 Network Security Visualization - Phase II Database Design Specification”, Applied Visions, Inc., 1998.

“SBIR AF97-043 Network Security Visualization - Phase II Data Visualization Specification”, Applied Visions, Inc., 1998.

“SBIR AF97-043 Network Security Visualization - Phase II User Interface Guideline”, Applied Visions, Inc., 1998.

“SBIR AF97-043 Network Security Visualization - Phase II Security Domain Specification”, Applied Visions, Inc., 1998.

“SBIR AF97-043 Network Security Visualization - Phase II Security Tools Survey”, Applied Visions, Inc., 1998.

“SBIR AF97-043 Network Security Visualization - Phase II System Architecture Specification”, Applied Visions, Inc., 1998.

“SBIR AF97-043 Network Security Visualization - Phase II System Requirements Specification”, Applied Visions, Inc., 1998.

The following progress reports have been delivered as part of this contract and serve as references for the contents of this report.

“SBIR AF97-043 Network Security Visualization - Phase II Status Report for April-June 1999”, Applied Visions, Inc., 1999.

"SBIR AF97-043 Network Security Visualization - Phase II Status Report for January-March 1999", Applied Visions, Inc., 1999.

"SBIR AF97-043 Network Security Visualization - Phase II Status Report for October-December 1998", Applied Visions, Inc., 1999.

"SBIR AF97-043 Network Security Visualization - Phase II Status Report for July-September 1998", Applied Visions, Inc., 1998.

"SBIR AF97-043 Network Security Visualization - Phase II Status Report for April-June 1998", Applied Visions, Inc., 1998.

The following serve as references for the contents of this report.

"Internet Scanner 5.2 User Guide for Windows NT", Internet Security Systems, Inc., 1998.

"SBIR Topic AF97-043 Network Security Visualization – Applications of VR to IW", Phase II SBIR Proposal, Applied Visions, Inc., 1997.

2 Status of Proposed Objectives

The overall goal of this Phase II SBIR was to develop a feature-full prototype of a software application that implements the concept of “network security data visualization”. The constructed prototype was intended to serve AFRL through integration with a system already having practical use to them and to create for AVI a technology with commercialization potential.

To achieve the Phase II goals, the following technical objectives were proposed and pursued.

2.1 Definition of System Architecture

Objective #1 was to define an extensible system architecture for the network security visualization prototype. Within the context of this contract’s research effort, this constituted defining a broad, product definition and a design for an architectural framework capable of meeting the defined system requirements. The results of these activities have been separately documented and delivered as CDRL A005R.

The *System Requirements Specification* (29-May-98) outlined the problems that NSV was intended to help solve, the environment in which it was expected to be deployed, who was expected to use it, how it was expected to be used and what features it would need to satisfy its users. From the beginning, it was not expected that every feature listed could be implemented within the scope of this contract, but a majority of them were addressed in some manner.

The *System Architecture Specification* (21-Aug-98) outlined the high-level architectural structure of the prototype system, the general manner in which the design met key requirements, candidate implementation technologies and the selected implementation technologies. However, the selected technologies identified in the document were superceded as a result of further product research and evaluation. Rather than using the OpenGL Optimizer 3D API, the WorldView VRML ActiveX control was used instead. Rather than developing custom distributed communications libraries, CORBA (specifically Visibroker) was used instead. These decisions will be discussed in more detail in later sections.

Objective #1 was achieved with the completion and delivery of the aforementioned documents, authored to a sufficient level of detail to support subsequent development efforts.

2.2 Definition of Security Data Domain

Objective #2 was to define a comprehensive domain of security data that the prototype’s database and visualization would have to support. Within the context of this contract’s research effort, this constituted conducting a survey of available commercial security products of various kinds and assembling a union of the data that they acquire. The results of these activities have been separately documented and delivered as CDRL A005R.

The *Security Tools Survey* (14-Jul-98) delineated a broad population of off-the-shelf network and computer security products. The population broke down into four general categories: (1) host attack simulators, (2) intrusion/packet monitors, (3) host integrity monitors, and (4) general-purpose network monitors. It must be noted that the survey constitutes a “snapshot” of a market that was, and still is, in a high degree of flux.

The *Security Domain Specification* (10-Aug-98) outlined the basic security data necessary for supporting software and database design and development. It intentionally did not list every known vulnerability and attack because there are always new ones being discovered and the prototype architecture was intended to be extensible in this respect.

Objective #2 was achieved with the completion and delivery of the aforementioned documents, authored to a sufficient level of detail to support subsequent development efforts.

2.3 Definition of Visual Idioms

Objective #3 was to define an effective set of 3-D visualization idioms and map the security data domain into them. Within the context of this contract's research effort, this constituted conducting a survey of data visualization "best practices" and then designing 3-D data presentations to be implemented by the prototype. The results of these activities have been separately documented and delivered as CDRL A006R.

The *User Interface Guideline* (14-Sep-98) introduced a set of guidelines for conceiving, designing and evaluating user interfaces in general and graphical data visualizations specifically. The document contained original ideas and information gathered from academic and commercial publications. The guidelines addressed topics such as design goals, visual cues, data organization, interactivity, scalability and evaluation methods.

The *Data Visualization Specification* (03-Nov-98) detailed the mechanisms and styles of security data visualization to be supported by the prototype. The overriding theme of the designs was highly configurable frameworks, such that a single scene design could present many different items from the security data domain. Within the frameworks were defined the details of how visual attributes would be used for various data items and their properties. In the end, three frameworks were designed: (1) the Association Scene, (2) the Scatter Scene, and (3) the Host Scene. Additional topics addressed by the design were actions, behaviors and scalability.

Objective #3 was achieved with the completion and delivery of the aforementioned documents, authored to a sufficient level of detail to support subsequent development efforts.

2.4 Implementation of System Architecture

Objective #4 was to design and implement the prototype's system software based on the research and specifications resulting from the preceding three objectives. The high-level designs resulting from these activities have been separately delivered as CDRL A006R, while the resulting program executables were delivered as CDRL A007R.

The *Database Design Specification* (26-Oct-98) defined the detailed database schema and the governing design conventions applied. The highly relational design contained 67 tables with rigorous foreign key relationships. In practice, the tables generally broke down into those that stored the acquired security data, those that supported security event type extensibility and those that supported general application features.

The *User Interface Specification* (10-Mar-99) defined the attributes, behavior and organization of the prototype's graphical user interface. Included in the design were an analysis of anticipated user tasks, the application's look-and-feel, the organization of commands and data, and two interface candidates.

The *Software Design Specification* (29-Mar-99) defined the general software component partitioning and the interfaces between them. At the next level of detail, it presented the object design of the software within several of the major components. Among the major components were the Client-Server Session, the Database Access, the Console Scene Construction and the Console GUI implementation. Two significant design aspects were the encapsulation of the 3-D viewer component and the use of CORBA as the distributed communication mechanism.

The final program executables from this objective consisted of the Console (for Windows NT) and the Server (for Java 2). The platform-independent Server arbitrated database access from the two client applications. The Console client application presented 3-D scenes to users. In its NSV configuration, the Server supports Microsoft Access97. (Although proposed, the development of a stimulator application was deemed superfluous and therefore forgone with agreement from the AFRL/IFBG.)

Objective #4 was achieved with the completion and delivery of the aforementioned documents, authored to a sufficient level of detail to support subsequent development efforts, and of the aforementioned program executables, demonstrably functional with respect to key requirements and capabilities.

2.5 Integration with Existing Security Monitoring Tool

Objective #5 was to integrate a third-party security "sensor" to effect automated data acquisition into the visualization system. Within the context of this contract's software development efforts, this constituted designing and implementing a so-called "data cartridge" application. The resulting program executable was delivered as CDRL A008R.

The final program executable from this objective consisted of the ISS Cartridge (for Windows NT). The Cartridge client application connects to the Server application to import into the NSV system database data that gets queried from ISS Internet Security Scanner 5.4.

Objective #5 was achieved with the completion and delivery of the aforementioned program executable, demonstrably functional with respect to key requirements and capabilities.

2.6 Integration with RL/IWT Expert System

Objective #6 was to integrate with the RL/IWT intrusion detection system to effect enhanced data visualization of its emerging network security technology. Within the context of this contract's software development efforts, this originally constituted designing and implementing a so-called "data cartridge" application. However, after consultation with the AFRL/IFGB, the agreed upon method was altered to integration of the Server application with a different relational database schema. As a consequence, the resulting program executables satisfying CDRL A009R, are in fact the same as those constituting CDRL A007R.

The final program executables from this objective consisted of modified versions of the Console (on Windows NT) and the Server (on Java 2). The platform-independent Server was altered to query the AIDE system's database, rather than NSV's native database. In its AIDE configuration, the Server supports the Oracle 7.3 RDBMS. The Console client was altered to construct 3-D scenes with the available information from the AIDE database. The Console changes were necessary because the AIDE database did not conform (in either form or content) with the underlying design conventions of NSV's native database.

Objective #6 was achieved with the completion and delivery on 01-Jun-99 of the aforementioned program executables to PRC (AIDE's vendor) for deployment during the August '99 AIDE demonstrations.

3 Technical Results from Prototype

This section summarizes key results of the effort to design and implement the network security visualization prototype system.

3.1 System Requirements

The *System Requirements Specification* was intended to formalize the vision of NSV as commercial product and as such included a broad collection of requirements, not all of which were directly related to the research topic of this contract. While this document served as a reference for the design and development of the NSV prototype and as many features as possible were implemented, it was recognized that not all of the delineated features could be fully developed within the allocated time and budget.

3.1.1 Implemented

The following is a broad list of proposed features that were totally or partially implemented by the delivered NSV prototype.

a) *Automatic categorization of detected vulnerabilities, intrusions and attacks* – Logically categorizes problems according to network service, affected host, responsible point-of-contact, etc. This is accomplished on the data acquisition side by the mechanisms used by Cartridges to upload their reportable security event types and on the 3-D side by the configurable attributes of the scene frameworks.

b) *Automatic prioritization of detected vulnerabilities, intrusions and attacks* – Assigns a severity level to each detected problem to support risk analysis and prioritized resolution. This is accomplished on the data acquisition side by the mechanisms used by Cartridges to upload their reportable security event types and on the 3-D side by the configurable attributes of the scene frameworks. Also, the relative ordering of event "severities" and "categories" can be adjusted from the Console display properties dialog.

c) *Presentation of network status and configuration* – Shows network hosts, active network services, etc., regardless of the presence or absence of vulnerabilities or attacks. This is accomplished on the data acquisition side by the mechanisms used by Cartridges to report actual event occurrences and on the 3-D side by the configurable attributes of the scene frameworks.

d) *Interactive display filtering* – Allows an operator to interactively adjust the suppression of display data based on problem severity, problem status, host attributes, network service, topology, point-of-contact, etc. in order to focus on specific areas of interest or responsibility. This is accomplished using "DataSets" to restrict the data assembled into a scene and using "Filter" commands to hide and show selected objects that are already in a scene.

e) *Pre-defined filters* – Provides pre-defined display filters, tailored to suit specific operator roles and tasks. This is accomplished by the ability to store/retrieve "DataSets" to/from disk. (Although the delivered software supports this capability, no pre-defined "DataSets" have been included on the delivered media.)

f) *Filter management* – Allows an operator to create, edit, save and delete customized filter definitions or templates. This is accomplished by the ability to store/retrieve "DataSets" to/from disk.

g) *3-D graphical data views* – Presents network security data using 3-D graphical renderings to increase operator cognition and efficiency. Uses various views to support operator tasks according to the previously identified roles. This is accomplished by configurable 3-D scene frameworks in the Console.

h) *Data view navigation* – Allows an operator to alter 3-D scene viewpoints and to traverse to related data views. This is accomplished by the 3-D viewer components inherent controls and by supporting mouse clicks to "drill in".

i) *Scalability* – Allows data presentation support for large numbers of hosts, vulnerabilities and attacks, and allow comprehensive, "big picture" analysis and "drill-down" access to lower-level details. This is accomplished by the scene design concept of "clustering" and the ability to enter them.

j) *Problem audit trail* – Maintains the bookkeeping associated with the detection, acknowledgement and resolution of security vulnerabilities and attacks. This is accomplished by the event audit database tables and their population by Cartridges and Console dialogs.

k) *Event acknowledgement* – Allows an operator to dismiss detected events that need no further action. This is accomplished by users changing the current audit state from a Console dialog.

l) *Automatic vulnerability resolution recognition* – Recognizes when an outstanding vulnerability has been resolved, removes it from display and records it in the audit trail. This is partially accomplished by the Cartridges reporting the absence of previously detected vulnerabilities. However, 3-D scenes do not currently change dynamically (see next Section) so users must request a scene refresh. Also, Cartridges cannot currently distinguish between not re-reporting a vulnerability versus not even looking for it.

m) Problem re-occurrence – Provides support for distinguishing between separate instances of a recurring vulnerability or attack. This is accomplished by the audit state sequence implemented by the Server's Cartridge interface.

n) Configurable severity – Allows an operator to override the default severity of any attack or vulnerability type to suit site preferences and priorities. This is a core capability of the system, but the event type dialog has not been fully implemented to expose it to users.

o) *Security descriptions* – Supplies an operator with descriptions of vulnerabilities and attacks. This is accomplished by the way Cartridges upload their reportable event types.

p) *Resolution recommendations* – Supplies an operator with recommended solutions for rectifying detected vulnerabilities. This is accomplished by the way Cartridges upload their reportable event types.

q) *Area of responsibility* – Allows an operator to assign hosts to user-defined categories to support filtering and partitioning based on organizational, functional, geographical, technological, etc., criteria. This is partially accomplished by the Console allowing host attribute assignments from a dialog. However, the Console currently does not support editing the lists of choices (which can be done by entering them directly into the database).

r) *Host criticality* – Allows an operator to assign a level of "mission-criticality" to each host (or "area of responsibility") to support filtering and partitioning. This is partially accomplished in the same way as (q) above. Also, the relative ordering of "criticalities" can be adjusted from the Console display properties dialog. However, although easily accomplished, "areas of responsibility" from (q) above do not currently support relative ordering.

s) *Operator login* – Enforces authorized, authenticated and audited operator sessions. This is accomplished by a mutual, encrypted, challenge/response connection sequence based on private license keys. The authenticated connections and operator logins are then logged in the database.

t) *Multiple operators* – Allows more than one operator to simultaneously interact with the system, each through a distinct instance of the user interface. This is a basic feature of the implemented system accomplished by Server side connection management. However, this feature may be subverted by what appears to be a bug in the CORBA library.

u) *Remote consoles* - Provides remote access from an operator console on other networked hosts. This is a basic feature of the implemented system accomplished by the CORBA infrastructure.

v) *Local console* – While remote consoles enhance certain operational aspects, during a denial-of-service attack on the network (as opposed to a specific host) only a console running on the same host as the server might reliably continue useful operation. A local console has the added benefit of not generating any observable network traffic of its own. This is a basic feature of the implemented system accomplished by the CORBA infrastructure.

w) *Multiple sensor types* – Allows support for, and integrates data from, each of the various sensor types (e.g., attack simulator, packet monitor, etc.). This is a basic feature of the implemented system accomplished by the extensible nature of Server's cartridge interface.

x) *Multiple sensor instances* – Allows support for, and integrates data from, multiple sensors, potentially including duplicates of a given type. This is a basic feature of the implemented system accomplished by the CORBA infrastructure and the extensible nature of Server's cartridge interface.

y) *Remote sensors* – Provides support for sensors that reside remotely on other networked hosts. This is a basic feature of the implemented system accomplished by the CORBA infrastructure.

z) *Local sensors* – Provides support for sensors that reside on the local NSV system host. A local sensor has the benefit of not generating any observable network traffic of its own. This is a basic feature of the implemented system accomplished by the CORBA infrastructure.

aa) *Extensibility* – Supports integration of new sensors in a modular fashion to avoid, or at least minimize, changes to the core system. This is a basic feature of the implemented system accomplished by the Server's cartridge interface which allows individual cartridge types to upload their own set of reportable event types and to register their presence with the system.

bb) *Cross-platform* – Allows the core system, the display console and the sensor interfaces to maintain cross-platform compatibility and inter-operability among Windows NT and Unix clients, and Windows NT and Unix servers. Cross Platform inter-operability was accomplished by the CORBA infrastructure. The Server can run on any platform for which a compatible Java 2 VM is available. The Console is restricted to Windows NT by design. The delivered Cartridge

only runs on NT, but the cartridge architecture supports development of Java-based cartridges for other platforms.

cc) *Minimized network bandwidth* – The system consumes as little network bandwidth as possible. Potential NSV network traffic includes server-to-console communications and sensor-to-server communications. Since the bulk of the network traffic between Console and Server is for 3-D scene constructions, network bandwidth is minimized by "clustering" and by transmitting ID numbers corresponding to property data values, which are then looked up in a client-side table. Since the bulk of the network traffic between Cartridge and Server is for reporting detected events, network bandwidth is reduced by pre-loading the reportable event types so that event occurrences can reference them by ID.

dd) *Network size* – Supports any number of hosts from 0 to 10,000. The upper limit has not been explicitly tested yet, but nothing about the system implementation explicitly limits the number of hosts. Support for a large number of hosts will certainly be affected by the capabilities of the available computer and RDBMS.

ee) *Heterogeneous network environment* – Supports security data reported on a heterogeneous mix of network hosts, to the extent provided by integrated sensors. There is nothing in the system design or implementation that limits the devices for which events can be reported.

ff) *Undiscovered hosts* – Accommodates "undiscovered" hosts residing outside an organization's network. Such hosts may be involved in benign sessions or malicious attacks. The system design only maintains an inventory of "targeted" hosts, which will essentially be those on one's own network. All other "source" hosts are only identified by IP within the event record itself, thereby imposing no specific limit on their number not already imposed by the number of events that can be practically supported.

3.1.2 Unimplemented

The following is a broad list of proposed features that were not implemented by the delivered NSV prototype. Although numerous, none were in fact required to meet the technical objectives of this contract. Rather, they were, and still are, considered useful features of a commercialized offering.

a) *Role-based authorization* – Establishes operator roles for the purpose of assigning levels of operator authorization. While the database supports the assignment of operators to roles, no operator permissions are maintained or enforced.

b) *Secure communications* – Given the sensitivity of the acquired security and network configuration data, a secured network connection inhibits sniffing and spoofing client-server communications. While encryption is currently used to authenticate NSV components when establishing communications, it is not yet used to encrypt normal message content. It should be noted that the data returned during scene construction consists almost entirely of cross-referenced ID numbers and not "raw data" as such. Also, the privacy of normal message content is less significant when the components are all running on the same machine.

c) *Secure database* – Given the sensitivity of the acquired security and network configuration data, security controls on the system database prevents unauthorized and unauthenticated access. This is really an RDBMS configuration issue, except for question of JDBC communications when the RDBMS and the NSV Server are running on different machines. This issue has not yet been addressed.

d) *View interactivity* – Provides 3-D views that dynamically change to reflect the current state of network hosts. While this capability has obvious desirability, it poses both logical and technical challenges that could not be addressed within the scope of the contract. The view may continuously change “out from under” the user because items would likely have to be repositioned spatially in the 3-D view in order to accommodate additional items. This could make it harder to examine a particular view of specific interest. From a technical perspective, the dynamic updating itself can be solved, and much of the system design currently accounts for it, but there were insufficient contract resources to complete it.

e) *Replay mode* – Allows an operator to replay historical events over a specified timeline. Given the relational nature of the data storage and the current mechanism used to retrieve scene data, “replay” mode presented a significant technical challenge that could not be addressed within the scope of the contract.

f) *Event triggers* – Allows an operator to receive special notification of specific events or status changes. While there are a number of “hooks” designed into the system architecture to support this capability, there were insufficient contract resources to complete it.

g) *Remote notifications* – Allows event triggers to invoke remote notifications to points-of-contact via beeper, pager, fax, email, IP socket, launched executables/scripts, etc. This capability would only have been worthwhile if (f) above was implemented.

h) *Data acquisition control* – Provides operators with some measure of control over the behavior of the available sensor interface modules and what information they detect and report. While there are a number of “hooks” built into the Cartridge interface with the Server, this capability has not been fully realized throughout the Cartridge implementation and the Console GUI.

i) *Automatic sensor correlation* – Automatically correlates events from multiple sensors so that a single event detected by more than one sensor would only be reported to the user once. This feature was intentionally removed from consideration because of its potential complexity.

j) *Event confidence* – Indicates a “confidence” level based on event correlation from multiple sensors. This capability only made sense if (i) above was implemented.

k) *Host identification* – Accommodates host renaming and re-assigned IP addresses. While the relational nature of the database data storage allows for easy manual renaming and IP re-assignments, there is currently no interface in the Console GUI for doing so, and there is nothing in the Cartridge implementation that can automatically detect such changes during the acquisition process.

l) *Mobile hosts* – Tolerates hosts that are only intermittently connected to the network. Nothing has been explicitly done in this regard, but intermittently connected hosts do not seem to pose any special problem for the current system implementation, provided they retain a consistent name and IP address.

m) *Network protocols* – Accommodates the effects of network protocols such as DHCP, WINS, and DNS. For the current system implementation, DHCP IP address re-assignment causes a problem in consistently associating events with hosts. However, it is expected that the current system architecture can support a solution by implementing another cartridge that monitors the DHCP controllers and reports reassignments to the Server.

n) *Database administration* – Offers some minimal set of internal database administration functions. Nothing has been done in this regard, since in many respects the DBMS can be effectively administered directly.

o) *Point-of-contact* – Allows assignment of a point-of-contact to each host, thereby designating the responsible party for maintaining it. Also allows a point-of-contact to be assigned to an entire “area of responsibility”, thereby becoming the point-of-contact for all member hosts not otherwise assigned. The current system implementation supports assigning NSV operators as points-of-contact, but this capability has not been included in the Console GUI.

p) *Task profiles* – Supports the definition of customizable profiles consisting of pre-set UI filters, data sets, reports, etc., to provide rapid user interface customization and switching in accordance with operator preferences and/or tactical work roles. The current system implementation supports assigning “profiles” to NSV operators, but neither this capability nor the RDBMS-based implementation of Scene/DataSet configurations has been included in the Console GUI.

q) *Smart sets* – Allows an operator to define inclusion criteria for customized, dynamic host groupings. They might be used for early warning, prioritizing work assignments, host culling, or display customization. This feature was never fully conceived and as a result was removed from consideration. In fact, its stated capabilities overlap with the current implementation of “DataSets”.

r) *Vulnerability reporting* – Allows an operator to “report” a vulnerability to a point-of-contact for resolution. . Without a complete implementation of (o) above, this feature was deferred.

s) *Web resources* – Integrates access to relevant Web resources, such as security advisories and hacker sites. This feature could not be addressed within the scope of the contract.

3.2 Prototype Implementation

Key attributes of the system architecture were achieved in the delivered prototype.

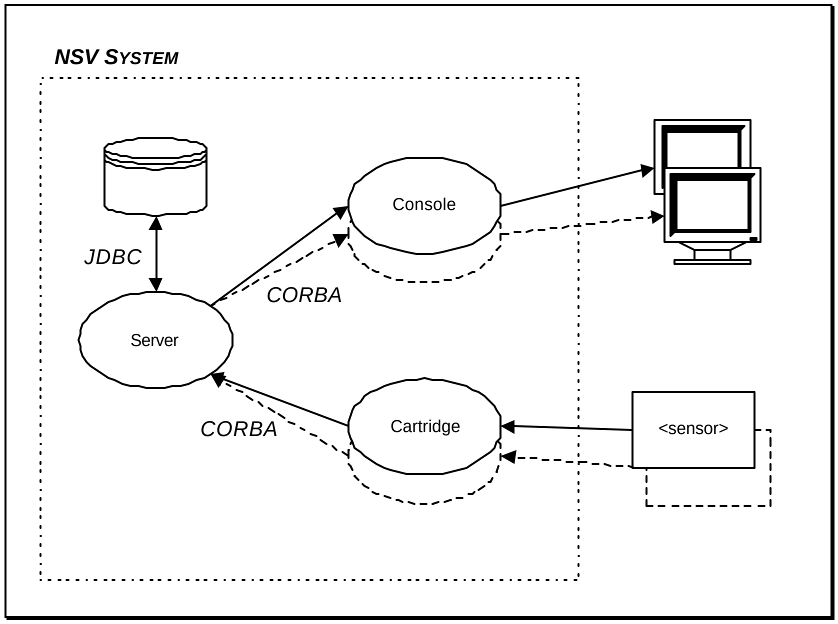

Figure 3-1 High-level Architecture

The system is "distributed" in the sense that the Console, Cartridge, Server and RDBMS can all reside on separate machines and can inter-operate across heterogeneous platforms. This will help facilitate the long-range goal of enterprise scalability. Already, the system is ready to support multiple Cartridges and multiple users running separate Consoles.

The system is "extensible" in the sense that it supports the addition of new Cartridges and new security event types, with no modifications to either the Server or the Console. Also, scene "frameworks" allow the 3-D visualizations to support additional data objects that might be later added into the database.

An important point to note is that the feature set for NSV's AIDE configuration has been essentially frozen since its delivery to PRC in May-June '99. As a result, there are effectively dual NSV implementations within the same prototype system, one based on the AIDE database and another based on NSV's native database design.

3.2.1 User Documentation

For details regarding NSV licensing, installation, configuration and known issues, see the *Installation Instructions* document, which is also included on the software distribution media as "`ReadMe.txt`".

For details regarding how to operate the user interface and how to interpret the 3-D scenes, see the *User Manual* document.

3.2.2 Console

The Console application was designed as a Windows application and coded in C++. Upon operator login, the Console connects to the Server application from which it will obtain security event data for visual presentation to the operator.

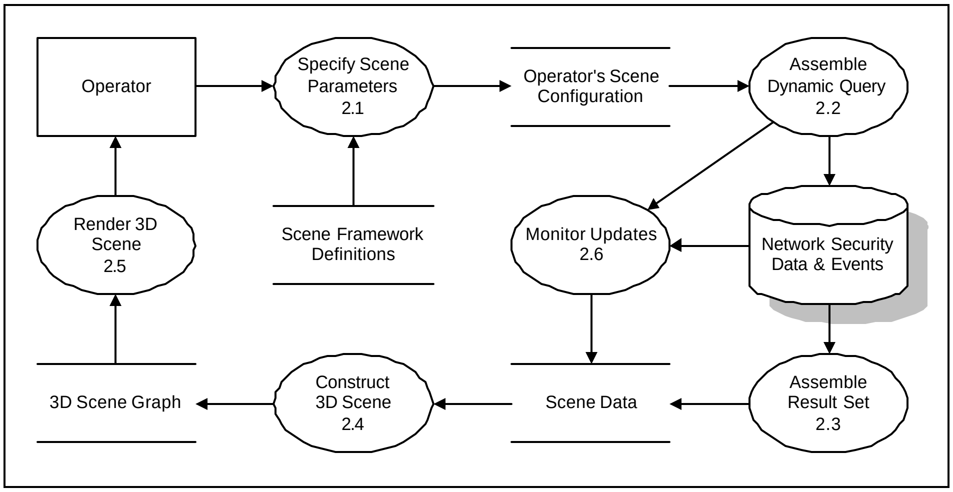

Figure 3-2 Scene Construction Data Flow

The Console consists of two executables. The `Console.exe` is the main application and consists of internal, logical subsystems for maintaining the GUI, accessing data from the Server and constructing scenes. The `Viewer.dll` is a separate library designed to support all of the scene building capabilities required by the `Console.exe`, but in a manner that isolates the details of the 3-D programming API. In this manner, it expected that other versions of the `Viewer.dll` can be implemented using alternate 3-D APIs. The current implementation uses a VRML browser ActiveX control.

The general design of the Console software can be found in the *Software Design Specification* document. Figure 3-2 summarizes the Console-Server processing with respect to scene construction.

3.2.2.1 Scene Design

The Console presents 3-D data visualization of security data using "scenes" designed to server as flexible analysis tools capable of comfortably and intuitively communicating a high level of data density. The scenes were designed to be scalable with respect to rendering time and comprehensibility. Each scene constitutes a "framework", so termed because each is capable of being driven by user-selectable data fields. In this manner, each scene design can yield thousands of distinct views of the acquired security data.

When this is combined with user-selectable "DataSets" (that allow data culling) and "filtering" of scene components, a simple yet powerful analysis capability results. The theories behind the scene designs can be found in the *Data Visualization Specification*, and the *User Manual* contains descriptions of how to select scenes, specify DataSets and work the filtering controls.

The scenes were implemented with a high fidelity to the designs, but there are two practical differences. The first is that the prototype system does not currently implement the so-called "inter-item associations" in the Association Scene. The second is that all Grid components work only on categorical data (or look-up tables) rather than as continuous scales.

3.2.2.2 Association Scene

The Association Scene is designed to show associations among specified data items and potential correlations among their properties. It can be said to communicate 10 “dimensions” of data through the use of item color, item size, item blink, item wiggle, item transparency, item placement in X, item placement in Z, associations from item up to category, associations from category down to item, and the chosen collection of items.

The example, show in Figure 3-3, is only one of thousands of potential configurations based on user choices of X property, Z property, item and category. Note that the other properties in use are pre-selected for simplicity.

3.2.2.3 Scatter Scene

The Scatter Scene is designed to show the distribution of the specified data items within a 3D cube. It can be said to communicate 9 “dimensions” of data through the use of item color, item size, item blink, item wiggle, item transparency, item placement in X, item placement in Y, item placement in Z, and the chosen collection of items.

The example, show in Figure 3-4, is only one of hundreds of thousands of potential configurations based on user choices of X property, Y property, Z property, color property, size property, blink property, transparency property, wiggle property and item.

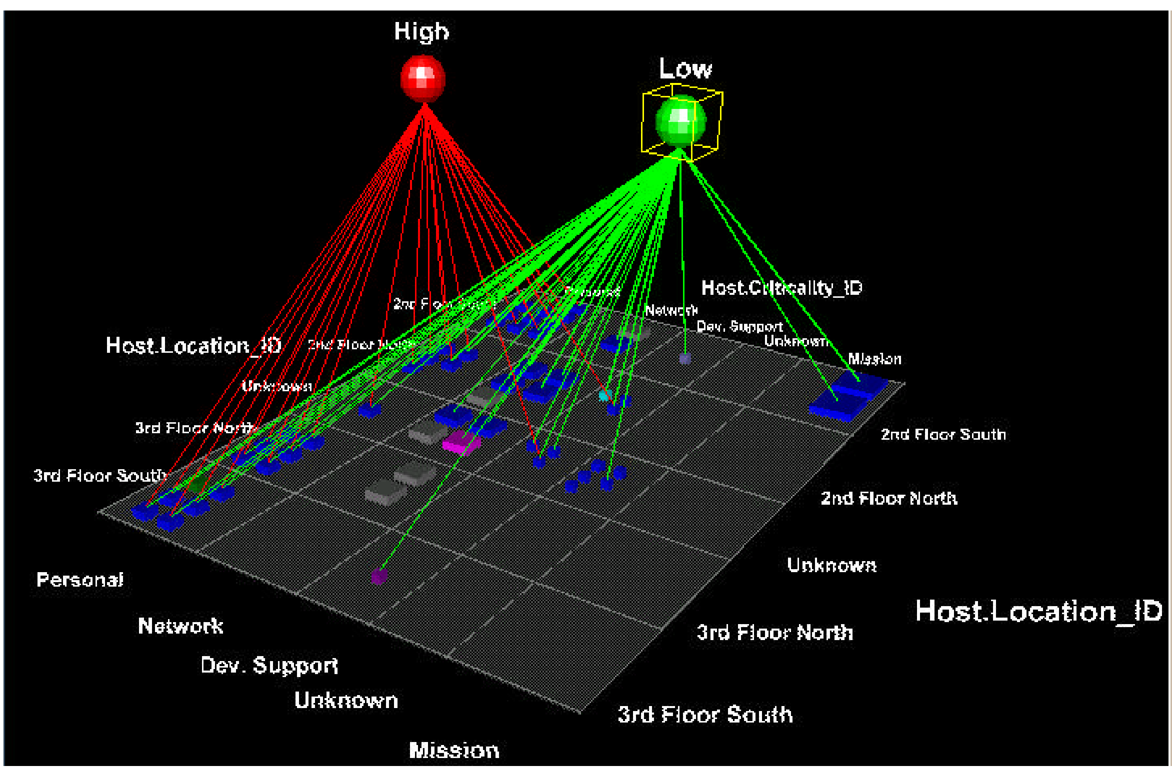

Figure 3-3 Association Scene

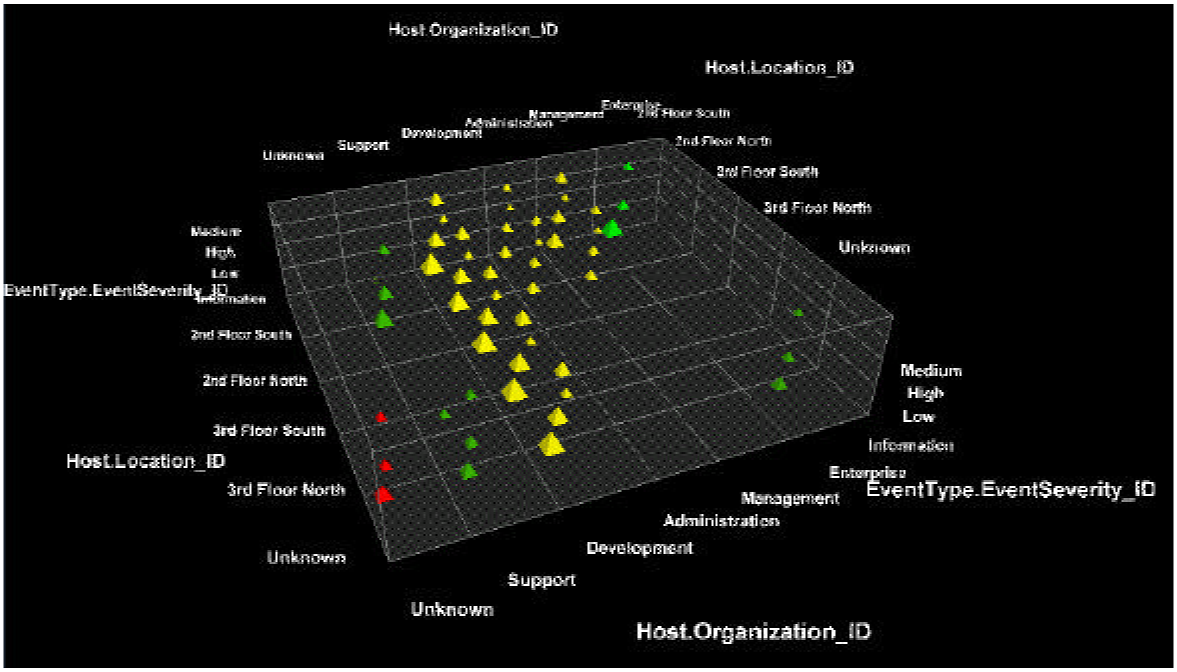

Figure 3-4 Scatter Scene

3.2.2.4 Host Scene

The Host Scene, shown in Figure 3-5, is designed to show resources, user accounts, network services, events, and associations among them for the selected host. It displays events classified as "transient" (e.g., attacks or suspicious activity) and events classified as "persistent" (e.g. vulnerabilities or inventory events) on separate grids, each sorted by event class and event audit status. Resources and User Accounts that are associated with the Host are displayed on separate grids, located in the back and at the bottom, respectively. Network Services associated with events detected on the Host are placed in a circular Category at the top of the scene. Associations represent any relationships that exist between the aforementioned objects.

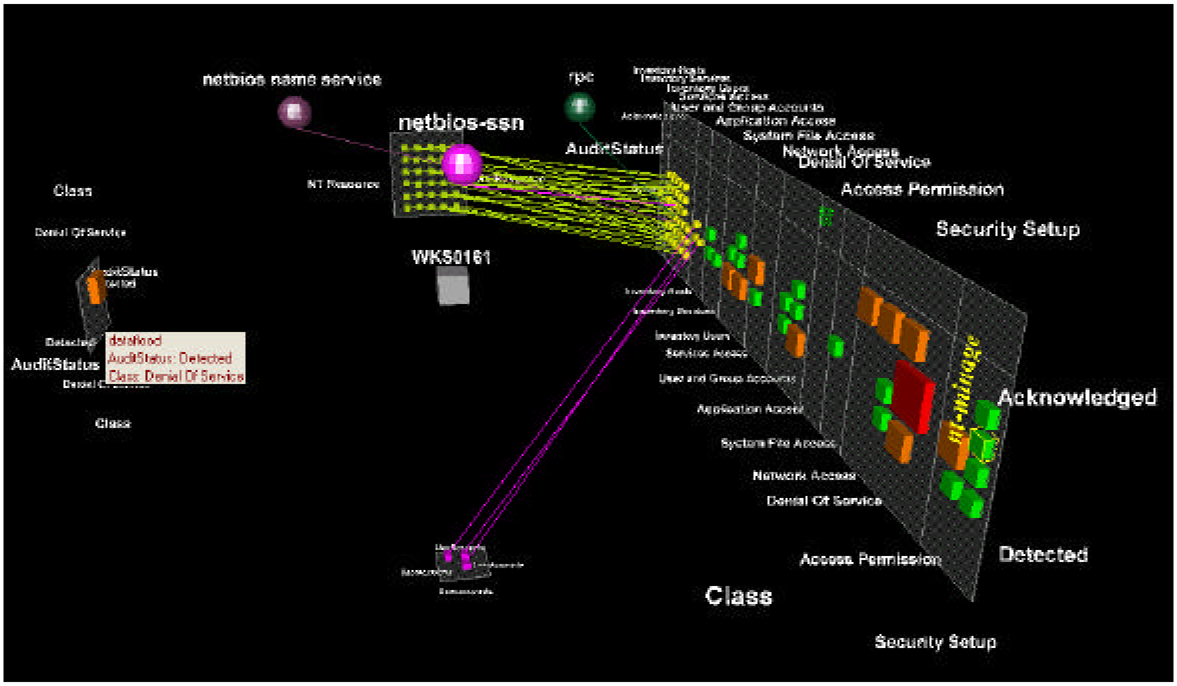

Figure 3-5 Host Scene

<u>3.2.2.5 Example</u>

The example shown in Figure 3-6 is an Association Scene of Hosts vs. Network Services. The data was generated by an ISS Internet Security Scanner scan of a live network and imported through a Cartridge component. The data was accessed through the Server component and displayed using the Console desktop application.

This Association Scene is simultaneously showing 10 "dimensions" of data, all visually accessible at a glance. A population (1) of Hosts are plotted on the grid according to their geographic Location (2) and their corporate Organization (3). Hosts are sized according to their Criticality (4) and colored according to their Operating System (5). Those that have any High Severity vulnerabilities (6) are blinking. No hosts have been attacked (7) since no items are wiggling. No hosts are “stale” (8) since no items are transparent. Individual Network Services are shown in a ring of bulbs above the grid and a connecting Association is shown when a Host is running a given Network Service. This scheme shows all Host running a given network Service (9) and all Network Services running on a given Host (10).

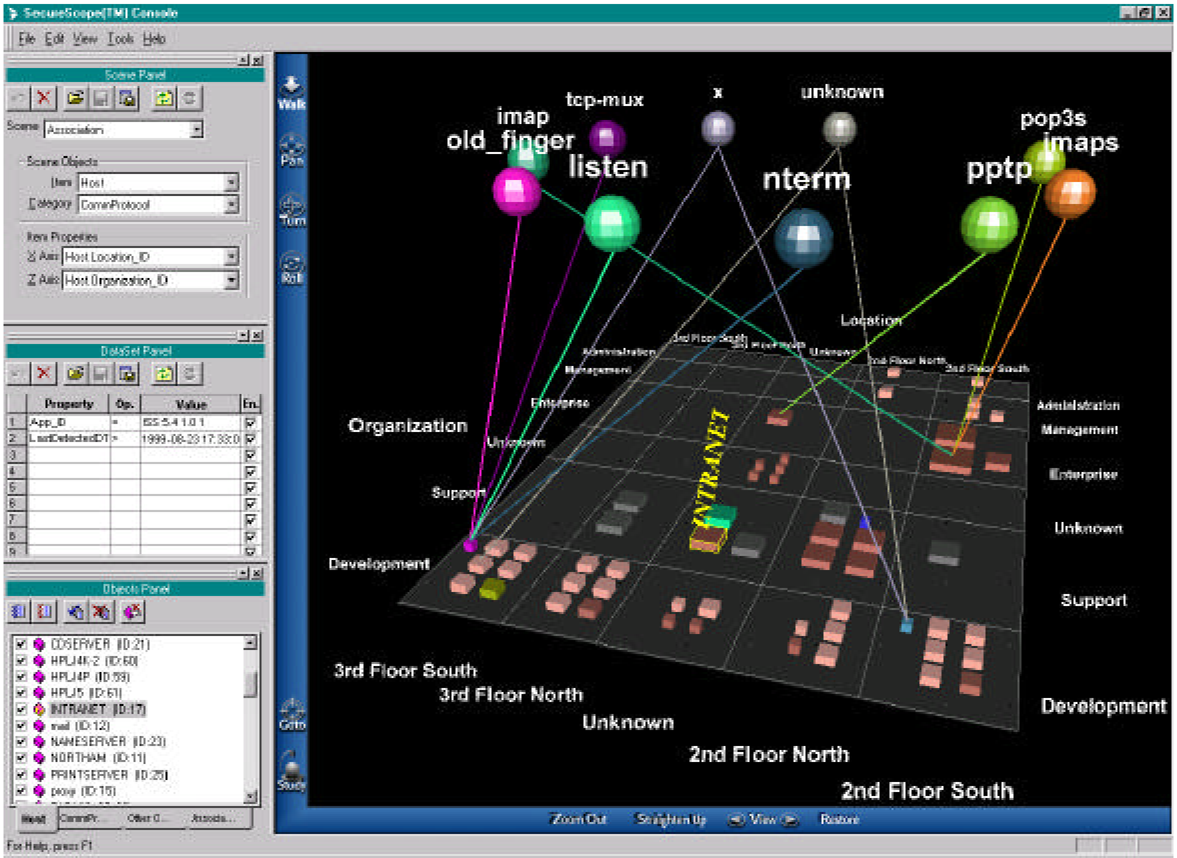

Figure 3-6 Console Screen Shot

The following lists a few points of information about the pictured network gleaned only from the scene shown.

- Only the enterprise mail server is running any mail protocols.
- Only two hosts (one Irix and one Solaris) are running X Window displays.
- There is only one host running Windows 95 and only one host running Windows NT 5.0.
- Only Windows NT 4.0 hosts have any High Severity vulnerabilities (i.e., blinking, but appearing as mid-blink-pink in the screen shot).
- Only two non-standard ("Unknown") Network Services are available on the network, namely the NSV Server running on a PC and Oracle running on the Solaris host.
- The two most critical (i.e., the largest) hosts are in the "Enterprise" organization and are located on the 2nd floor in the south wing.
- It also appears that "Development" hosts are evenly distributed throughout a building, whereas "Management" and "Administration" hosts are concentrated on the 2nd floor.
- The highlighted "Intranet" host does not show HTTP running because the HTTP Service was filtered out of the display (note, no bulb) using the Objects Panel.

3.2.3 Server

The Server application was designed as a platform-independent application and coded in Java. It uses several threads executing within a single instance of a Java Virtual Machine. It consists of internal, logical subsystems for accepting client connections, maintaining database connections, performing SQL generation and result-set binding, inserting acquired security events into the database and gathering the requested data for Console scene construction.

The general design of the Server software can be found in the *Software Design Specification* document.

3.2.3.1 Table Access

As detailed in the *Software Design Specification,* all of the Server code that implements table-specific access to database information is auto-generated based on the database schema. Furthermore, all of the code on both the C++ client and Java Server that "remotes" this capability by wrapping it into CORBA objects is also auto-generated by a VBA script.

Built into the auto-generated table access objects are the preferred join paths between tables. This information is encoded as tags in the tables' descriptions for subsequent parsing. These preferred join paths define the 'meaning' of the underlying associations between tables, which are described in the *User Manual.* These paths are used for relating data from multiple tables during scene construction and "DataSet" processing.

Using the auto-generated table access objects and relying on their internal join path tables, client applications (and much of the Server itself) never have to deal with SQL directly. This is one aspect of laying the groundwork for supporting RDBMSs from multiple vendors.

3.2.3.2 Scene Data Querying

Special scene data objects (one per scene framework) handle the complex process of acquiring all of the necessary data for constructing a given scene. These objects select data from multiple tables and so they do not use the auto-generated table access code exclusively. Instead, they generate SQL based on the user's scene settings to gather the bulk of the scene data.

In order to implement the scene item "clustering" (which aids scene scalability), preliminary queries are submitted to determine whether clustering is necessary. If not, "item" queries are performed to select all of the required scene data. Otherwise, "cluster" queries are performed with SQL grouping and aggregate functions in order to yield clusters rather than individual items.

3.2.4 Cartridge

The Cartridge application was designed as a Windows command-line program and coded in C++. It was developed to interface with ISS Internet Security Scanner 5.4. The Cartridge consists of one executable. The `ISSCartridge.exe` is the main program and re-uses the client-server connection code from the Console. A Cartridge consists of a sensor-specific "translator" class that inherits from a "base" translator implementation. This minimizes the amount of new code development for each new Cartridge. To create a new Cartridge for another sensor, only the translator sub-class would have to be written.

3.2.4.1 Operation

Upon start-up, the Cartridge connects to the Server application and goes through a registration process that involves a one-time upload of its population of event types that it might later detect and report. From this point forward, the Cartridge need only supply the information unique to the event detection and merely reference the previously uploaded event type. This mechanism is designed to provide system extensibility by allowing Cartridges to define their own event types. It allows other Cartridges to be implemented to report new event types with no changes to the Console or Server applications.

After registering, the Cartridge downloads the collection of events that it previously reported and that are still "alive". This is done so that it can detect the current absence of previously detected network assets and vulnerabilities. When an ISS 'job' completes, the Cartridge reads the data from ISS's Access database and submits a collection of normalized, detected events to the Server, which is then responsible for updating the database.

The following data flow diagram summarizes the Cartridge-Server processing with respect to acquiring security data from remote, third-party sensors.

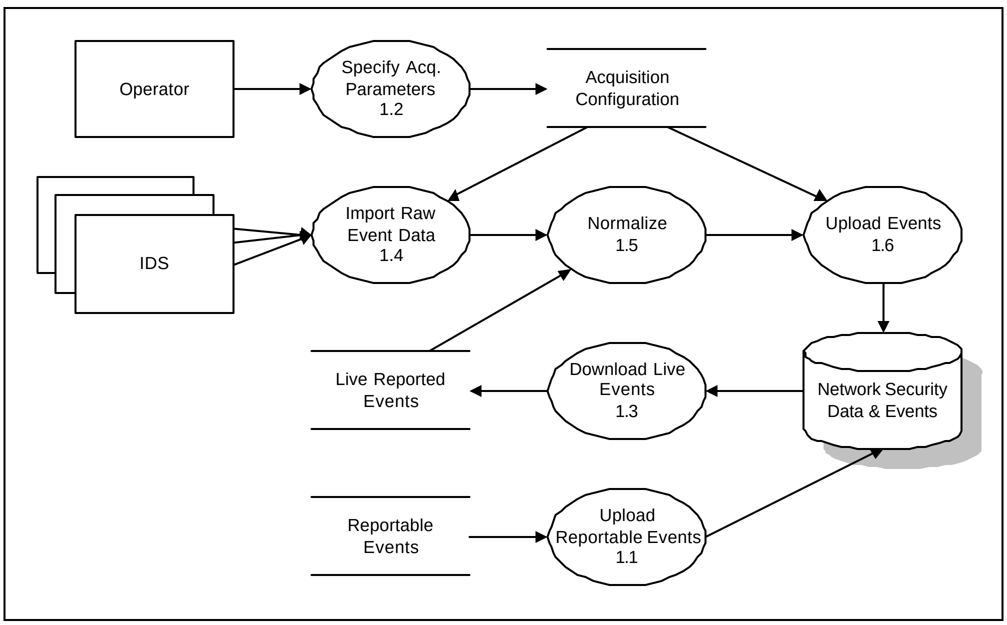

Figure 3-7 Acquisition Data Flow

3.2.5 Lines of Code

The following tables summarize the amount of code that makes up the completed prototype system. The "authored" code represents all of the software written by AVI's software developers. The "auto-generated" code represents all of the software created by a code automation tool, which was itself authored in VBA by AVI's software developers.

Language	Files	Total Lines (w/ comments)	Lines of Code
Java	55	27,800	13,000
C++	338	90,400	40,800
CORBA IDL	17	2,100	1,200
VBA	3	5800	5500
VRML	12	1,200	1,200
Total	**425**	**127,300**	**61,700**

Table 3-1 Authored Code

Language	Files	Total Lines (w/ comments)	Lines of Code
Java	163	68,600	37,700
C++	110	24,100	10,600
CORBA IDL	52	7,500	3,400
Total	**325**	**100,200**	**51,700**

Table 3-2 Auto-generated Code

3.3 Evaluation

The overarching goal of NSV is to be able to rapidly comprehend the overall network status (i.e., the "big picture") and yet be able to flexibly narrow the focus to targeted areas. From working with the final system of visual data presentations of data acquired from a live network, it appears that this goal is indeed served by the visualization concepts developed under this SBIR. In this regard, and with recognition of several areas for improvement, the prototype system developed under this SBIR is offered as validation.

The following sections present selected evaluations of key NSV developments.

3.3.1 Data Abstraction

Abstract visual idioms were intentionally chosen to represent detailed data values in order to allow the 'eye' to comprehend a whole population of items, rather than force a user to read the information and formulate some kind of mental 'picture' from the words. The scenes in the prototype system were in fact implemented with a high degree of fidelity to the designs in the *Data Visualization Specification*. This has resulted in the desired levels of data integration and density with scenes.

This methodology does, as expected, allow for the visual discovery of patterns, trends and correlations among disparate data properties and items. However, the abstract nature of the presentation, by design, pushes down the discursive descriptions of the states of individual items to a level further from direct user access than certain conventional user interfaces. In this respect, it may be perceived by some as a 'double-edged sword'.

3.3.2 Scene Frameworks

The concept of scene "frameworks" proved highly beneficial in two key respects. First, they highly leveraged the scene development process such that a single scene design could be used to present a wide variety of data. This allows focused data views without having to develop new scenes for every potentially useful combination of data properties. Secondly, they make maximum use (and re-use) of simple visual attributes and structures in order to minimize the learning curve related to understanding the abstract presentations by leveraging the familiarity of common themes, such as grids, associations and clusters.

However, the existing scene frameworks do not work well with data properties that contain a great many unique property values, as is the case with time fields and IP addresses, for example. This argues for the development of another framework with different mechanisms that address this special need.

Also, while it is believed the second benefit above has been realized, owing to the highly configurable nature of the scene frameworks, it may yet be confusing to users the actual meaning of a particular visual attribute at any given time. For example, in the Association Scene, color is used to indicate the operating system of Host items, but for Event items it indicates the event's severity. This is a perceived downside to the generic (but flexible) presentation techniques used by the designed frameworks. It is expected that offering a legend will allow the user to remain "oriented" with the meaning of the various visual attributes in each scene. This was, in fact, recommended by the *User Interface Guidelines* document, but was regrettably not accounted for by the actual design captured in the *User Interface Specification*.

3.3.3 Clustering

Clustering has proven to be an effective scalability mechanism. It reduces the data traffic between Sever and Console, it reduces the number of scene objects that have to be added into the viewer component, and it maintains a useful amount of information while drastically reducing scene clutter and navigation difficulties.

However, the virtual necessity for users to specify additional configuration info in order to effectively drill in to a cluster can be inconvenient. Moreover, if items do not have a wide distribution of values for several data properties, it may be difficult for a user to effectively 'slice up' the data to a sufficient extent to get below the clustering threshold. See the "suggestions" section below for proposed ways of easing this burden.

3.3.4 Scene Data Querying

The Server generates at runtime the SQL required to query for scene data. It first determines whether or not clustering is necessary and then adjusts the subsequent querying accordingly. When clustering is necessary, care is taken to ensure accurate counts of items within clusters. In either

case, outer joins are required in nearly all aspects of querying for scene data to ensure that complete item populations are not artificially truncated, either when fields contain NULLs or when potential property values do not happen to be currently in use.

Owing to difficulties in formulating a single cluster query and a single non-cluster query, a sequence of queries is used to determine whether or not clustering is necessary, what the cluster counts are, which associations exist and what the full list of potential scene property values are. This was partly the result of the complexity of the process, but also partly a result of limitations and/or differences among SQL implementations from different vendors.

The result in the current version of the prototype is that queries could likely benefit from some further analysis to optimize their performance. Also, outer joins are not yet fully implemented when using MS-Access as the RDBMS. Doing so is not an insurmountable technical difficulty (and was indeed implemented for Oracle databases), but to the extent that different RDBMSs require different SQL to accomplish this, it becomes an awkward maintenance task when seeking to support databases from other vendors.

3.3.5 Pre-selected Join Paths

The ability to capture the relationships between tables in the schema design and automatically generate the relevant join paths and the code to make use of them has proven very efficient to maintain and alter during development. Moreover, using a single, pre-established relationship between each pair of tables ensures a predictable ‘meaning’ to the data presentations and ultimately reduces the perceived complexity of the application.

However, having to choose only one join path between each pair of tables is restrictive when tables are directly or indirectly related in more than one way for useful reasons. This means that alternate ‘meanings’, although potentially interesting, are not accessible to users. Also, in the case of the NSV database schema, where the join paths are generated automatically, this issue affected the actual schema design, wherein circular relationships had to be removed. Assigning join path tags to individual tables has to be done very carefully to avoid introducing unsupported circular relationships.

3.3.6 Undetected Events

The current cartridge implementation is capable of reporting “undetected” events (e.g., hosts missing, vulnerabilities corrected, etc.). This requirement derives from the goal of maintaining an accurate inventory of network assets and their configurations as opposed to collecting isolated snapshots of events. However, the cartridge currently relies on ‘batch’ reporting of events and assumes that it always operates using the exact same sensor configuration.

One as-yet-unrealized goal of the cartridge data acquisition process is to be able to tell the difference between something being not found because it no longer exists on the network as opposed to being not found because it was not even looked for. For example, the cartridge does not currently know whether to report that a vulnerability is gone from a host (perhaps because it was corrected) or whether to report nothing because the scanner was configured to not even look for it.

3.3.7 Client Programming Language

The system architecture and the Server implementation inherently support clients running on heterogeneous platforms, written in different programming languages. Currently, the only clients in use were written in C++.

In order to support Java client applications (e.g., platform independent cartridges) the client-side application session software and remote database access classes must be ported from C++ to Java. This is not expected to be a large task since the hand-authored session code is relatively small and the remote database access code is auto-generated. However, in the case of cartridges, the base implementation will also have to be ported. But, these efforts do not involve any changes to current clients or the Server because the system architecture was designed with this eventuality in mind.

3.3.8 WorldView

WordView was chosen as the 3-D rendering component because its advertised capabilities matched closely with those required by the NSV application and it would integrate seamlessly into the application framework of the Console. Indeed, during development these perceived benefits were born out.

However, the more WorldView was stressed by the growing functionality of the Console application, the more flaws became apparent. WorldView requires very large amounts of memory and it hoards the CPU. Much worse, it exhibits severe memory leaks and sometimes crashes for unknown reasons. It being an ActiveX component, developers have no visibility into its internal workings and therefore cannot research and correct its apparent problems. A great deal of time, money and effort has been expended developing workarounds for what are believed to be WorldView's problems.

Significantly, since deciding to use WorldView, it has become an unsupported product. As a result, no upgrades are available so no internal bugs are getting fixed. In the end, it would seem that taking NSV further may require replacing WorldView. This is a significant issue but not catastrophic. On the one hand, NSV can continue in reasonable fashion with WorldView as it is. On the other hand, the 3-D viewer was designed into a separate DLL so that the Console can work unchanged with another 3-D technology provided a new DLL is developed.

3.3.9 Visibroker/CORBA

The capabilities of CORBA in general have proven their value and have been beneficial to the system design and development. The prototype system makes use of its platform interoperability, local and distributed communications, and multiple programming language bindings. It has conveniently obviated the need to write significant amounts low-level socket code and message marshalling and decoding.

However, problems were encountered with the Visibroker-specific implementation libraries. Visibroker has exhibited some flaky behaviors and it apparently leaks memory during data marshalling. In general, problems could not be duplicated using simple examples and yet there were no apparent errors or even differences in usage between what worked and failed. Moreover, Inprise charges a significant runtime license fee per deployment that has turned out to be unexpectedly burdensome.

One difficulty with the CORBA IDL mapping to C++ is that modules are mapped to nested classes, meaning that all CORBA interfaces ultimately wind up defined within a single C++ class. This turns trivial IDL changes into extremely time-consuming C++ re-compiles since all code is essentially dependent on a single, very large include file. No such problem exists with the IDL mapping to Java because modules are mapped to separate ‘packages’.

3.4 User Feedback

One technical objective of the contract was to integrate with AFRL’s AIDE/EPIC system. This was accomplished in the accelerated timeframe requested by AFRL, but NSV was not deployed by AIDE’s vendor to users as part of the official AIDE demonstrations. As a result, the opportunity to gather objective feedback from the AIDE user community was effectively lost.

However, the following comments were offered by users at an EPIC site and by AIDE users at a training session conducted by AVI.

- Allow the scene configuration choices of tables and fields to be altered by users. [This could allow changes to the AIDE-EPIC DB schema without requiring source code changes to NSV.] “*The problem with the static configuration is that the user community that we serve often cannot conceptualize or describe what they want to visualize until they see it. This flexibility affords us the opportunity to ‘fine-tune’ the visual displays to provide the information that the security analyst needs to see to describe the information assurance situation.*”
- Allow the DataSet Panel to support a variable number of criteria, beyond the current restriction of ten.
- Add the ability to print the scene and optionally the scene configuration information.
- Allow configurable fonts and point sizes for text labels in the scene. “*This helps me avoid zooming-in and zooming-out repetitively to see* [small] *labels. It's also helpful for printouts.*”
- Ease the user entry of DataSet criteria values by using formatted controls or point-and-click controls to avoid having to type in each value literally. [This comment pertains to version 0.1 delivered for early use with AIDE. This issue has been partially addressed in the current version 0.2 through the use of pick lists and a date/time control.]
- Allow a scene to be ‘saved’ for later recall. [To suit different purposes this could be accommodated (1) by saving scene data for later import into NSV or (2) by exporting VRML for use in generic web browsers with a VRML plug-in.]
- Allow creation of ‘executive’ reports with summary statistics. [This feature appeared in the *System Requirements Specification* but is as yet unimplemented.]
- Add a ‘replay’ mode capable of animating the time-sequence of events. [This feature appeared in the *System Requirements Specification* but is as yet unimplemented.]

3.5 Suggested Improvements

Evaluating the completed prototype system, the following features are suggested for improvement.

a) *Scene attribute legends* – A Legend Panel would allow users to see the meaning within the current scene of object attributes such as size, color, wiggle, and blink. Given the highly configurable nature of the scene frameworks, it may not be clear what a certain colored box means, or what it means when it is blinking. Therefore, the ability to display this information in a collection of legends is important.

b) *Forward/backward scene navigation* – This capability is present in the *User Interface Design* document, but there were insufficient contract resources to implement it. Not being able to back up to a previous scene has (as expected) proven awkward. This is especially true after traversing into a Host scene from an Association Scene or Scatter Scene because many previous settings get tossed away. This feature would also effect "drill-outs" after having drilled in.

c) *Item attribute filtering* – This feature would significantly extend the current filtering capability. Currently, filtering only allows scene objects to be hidden/shown by specific selection in the Objects Panel. The ability to filter on item attributes would allow all items with a selected attribute value to be hidden/shown. To avoid having to re-query the Server database, this filtering would operate on whatever item attributes are present in the scene as a result of its configuration settings. For example, a user might go to the Legends Panel and select any combination of available choices for X, Z, color, size, blink and wiggle property values.

d) *Auto-select axis properties on cluster drill-in* – This feature would allow users to avoid having to manually specify new axis parameters when drilling into a cluster. This is proposed because it is sometimes burdensome to have to make unique selections in the drill-in pop-up dialog. Making unique selections is required because using redundant choices will not expand the grid axes, which in turn will not "dice" up the original cluster. In this case, the result would be a scene with only one cluster; specifically, the single cluster already clicked upon. On the other hand, auto-selection would allow the user to choose not to see the drill-in pop-up dialog at all and the system would select axis properties that have not yet been used, resulting in a simplification for the user. This user could configure preferences for which choices to apply earliest, or turn off the feature entirely.

e) *Last chance cluster dicing* – This feature would be employed to address the difficulty that arises when a cluster cannot be sufficiently "diced" by grid axis property selection to allow drilling in far enough to get below the ClusterThreshold. When employed, it could manufacture sub-clusters based on such things as associations with categories, item ID ranges and/or time ranges.

f) *"Other" drill-in selection* – This would allow users to choose which group of "Other" categories or items to expand, rather than sequentially viewing each of them. When the number of categories in a scene exceeds the configured CategoryThreshold, only the most populous ones are displayed individually and the rest get grouped into an "Other". In order to see these other categories, the user clicks on the "Other" scene object and the next most populous ones are displayed. When a user is interested in a certain category, displaying it may require cycling through several groups. This can be improved by having a pop-up dialog present the groupings of non-displayed categories so that the user can directly select the one group with the desired category. The behavior is analogous for "Other" item clusters that appear when the number of bins in a grid axis exceeds the PropertyThreshold.

g) *Scene with time theme* – This would give users a better picture of the time sequence of event occurrences. The existing scene designs do not explicitly take times into account (although DataSets can be used to restrict scene content to specified times). A scene specifically designed to show time would depict the relative timing and sequencing of detected events and audit state changes.

h) *GUI upgrades* – There are many aspects of the Console GUI that are currently incomplete. Specifically, the following items should be added: the Workspace Panel w/ drag-drop; more detail dialogs for scene objects; online help; property dialogs for adding/editing/removing "organizations", "locations", "criticalities", "severities", etc.; management dialogs for editing the assignment of IP addresses to Hosts, for de-activating event types, etc.

i) *Database table consolidation* – An improvement in database query performance should result from consolidating property tables into the existing display properties table. It has turned out that so many of the property tables are so similar that the tables could likely be removed by moving their fields into a single pre-existing table.

j) *Outer joins with MS-Access* – Owing to the differences in SQL syntax between Oracle and MS-Access, so-called outer joins are currently not supported when using MS-Access. Addressing this will correct situations where scene content may be under-reported when it is not specifically related to chosen data properties. This affects clustered Associations Scenes, when Item Clusters have no Associations to Categories, and Association Scenes and Scatter Scenes, when Items have a "`NULL`" value instead of a reference to a known Axis property.

k) *AIDE configuration upgrades* – Currently, NSV in the AIDE configuration does not support the Host Scene or the Scatter Scene.

l) *Replace CORBA* – While generic CORBA offers the necessary and desired functionality for distributed communications in NSV, difficulties with vendor implementations suggest important benefits from replacing it with a custom communications implementation. Specifically, eliminating Visibroker should eliminate high runtime licensing fees and the inability to find and correct apparent bugs within the ORB implementations.

m) *Replace WorldView* – While the WorldView ActiveX control advertises the necessary and desired functionality, it has become unsupported by its vendor, it has internal memory leaks and apparent synchronization problems, its scene build/destroy performance is marginal, and its rendering has some unpleasantness. It seems that the only way to further optimize scene build times and gain control over resource and synchronization issues is to develop a custom viewer window based on lower-level 3-D drawing APIs.

n) *Unimplemented features* – There are many proposed features listed in the previous sections.

o) *Variable number of DataSet Criteria* – Currently, DataSets are limited by the GUI implementation to ten criteria. Although criteria can add to the total query time (and therefore apparent scene building time), there is no inherent reason for enforcing a limit of ten. Having a fixed number of criteria can be overly restrictive inasmuch as they are also used to implement cluster drill-ins.

p) *Add more operators in DataSet criteria* – Although nulls in the database can currently be tested in DataSets by combining the '=' or '<>' operators with the "NULL" value keyword, this does not work with value types that utilize certain controls to data value entry. For example,

there is currently no way to test nulls in date fields because the date/time control does not allow a user to enter the "NULL" keyword. Adding "Is Null" and "Is Not Null" operators could resolve this situation. Also, it would be desirable to allow "Not Like" in addition to "Like". There is currently no way to specify whether a criteria should be logically 'AND'd or 'OR'd, but it is not clear whether this additional capability is worth the additional complexity for users.

q) *Abort Server-side querying* – Since database queries may take a long time, it would be desirable to be able to abort them. Currently, only client-side scene construction can be aborted.

4 Addenda to Referenced Specifications

This section outlines differences between certain referenced technical reports and the delivered prototype executables.

4.1 System Architecture

4.1.1 Features

The Console does not cache security data as proposed. At design time, this was deemed costly to develop and ultimately impractical owing the potential volume of data and the highly flexible nature of the scene frameworks.

While the concept of “Gateways” still applies, none were authored under this contract since they were not part of meeting the contract’s technical objectives.

Regarding client-server message communications, clients do not “register” with the Server for messages of interest, as proposed. Instead, an architecture of distributed “objects” allows distributed components to invoke remote methods to effect system behaviors. By design, all remote Server objects have corresponding client-side callback objects that allow the Server to initiate communication with clients.

Asynchronous messaging is not a feature of the implemented system because it was deemed largely unnecessary and because it can increase system complexity by increasing synchronization complexity.

Store-and-forward capability is not a feature of the implemented system because it requires an asynchronous messaging mechanism, it was deemed unnecessary for the Console interface to the Server, and the Cartridge interface to the Server could be extended to achieve similar benefits.

Server fail-over was not within the scope of the contract or technical objectives, and is therefore not a capability of the delivered prototype.

4.1.2 Selected Technologies

The Console was indeed written for Windows NT, based on Win32/MFC APIs. However, after an evaluation of the practicality of porting it (along with its interface to the 3-D API) to UNIX and of projected market demands, this idea was abandoned as ultimately not worth the effort.

Java 2 was chosen for the Server component to ensure its portability among UNIX and NT server platforms. Although its generally poorer performance was a consideration, most of the Server’s work is performed within its RDBMS.

The implemented Cartridge was written for Windows NT to leverage code commonality with the Console, but the implemented system architecture will inherently support Java cartridges too.

CORBA was ultimately chosen to provide cross platform, distributed communications, rather than authoring a custom-built library. The arguments against CORBA were that it was “heavy weight”, it unnecessarily added to technical complexity, and its implementations had incomplete services. However, it was determined that a subset of its technical capabilities (without using any of its services) could be engaged to eliminate the complexity and that performance of large transfers was

satisfactory. Its benefits were (1) the elimination of the necessity to code socket message marshalling and decoding and (2) its object-oriented architecture of remote method calls. This was further simplified through auto-generation of significant portions of database remote access code.

WorldView was ultimately chosen as the 3-D API. We were unaware of WorldView at the time the spec was written. It was chosen for its easy integration as an ActiveX control into a Windows GUI, for it's high level programmatic interface and for its support of VRML. These served to reduce complexity and leverage in-house knowledge of VRML. Furthermore, its support of DirectX made its performance better on most common PCs.

4.2 Database Design

The delivered system implements about 80% of the defined tables. Some tables were left out because their purpose had been superceded or eliminated. Others were made obsolete owing to restructuring designed to reduce the length of join paths and to eliminate the potential for circular join paths. The latter became an issue when an automation tool was used to automatically generate relevant join paths between tables by parsing the relationships in the database schema. This was done as part of the auto-generated database access layer implementation.

Minor field changes were made to several tables to support Cartridge and Console functions, and to better reflect the desired security data domain representation.

4.3 User Interface Design

The Filter Panel was reconsidered because the design would have required more database querying and, as a result, would have been less responsive to a user. A rudimentary form of client-side filtering was added to the Objects Panel to hide/show specific scene objects or whole classes of scene objects. This operates instantaneously because no requests to the Server or database are required. Although not as capable as the original design, (1) it meets the primary goal of Filtering, which is to reduce scene clutter in order to focus in on specific scene objects, (2) it can be made more capable with straightforward additions, and (3) the effect of the original design can yet be achieved using so-called "DataSets".

Development of the Workspace Panel was deferred in order to meet AIDE requirements and schedule. Instead of a database-oriented implementation of a separate panel of workspace objects (like persisted Scene and "DataSet" configurations), a simple file-based storage implementation was used in Scene and "DataSet" Panels directly. This had certain effects on usability but provided a core capability with minimal fuss.

4.4 Data Visualization

Host scene design does not use "iconic" representations as its primary design theme, as proposed. Instead, it is based on the same abstractions as the Association Scene framework. This has the benefit of leveraging the components used by other scene formations. It also has extended the Host Scene's potential because its client-side implementation can be used as another generic framework, like the other scenes. As a result, it could be made user configurable and/or it could support data items besides hosts. The design seeks to present the same collections of host data as that shown in the spec, but does so in a way the can communicate more information. It does so by eliminating the literal-looking texture mapping and replacing it with objects for which color, size, placement, motion

and blink represent separate data properties. Of equal importance is the addition of associations between related items within the scene.

5 Year 2000 Compliance

The Year 2000 Problem (or Y2K Problem) results from the incorrect handling of date information by computer-based systems. Computer-based systems often store dates only two-digit years. The year 2000, therefore, may mistakenly be interpreted as the year 1900. This will result in incorrect date comparisons causing date sensitive operations to fail. Computer-based systems may also fail to recognize the year 2000 as a leap year, which contains 366 days as opposed to the normal 365 days. Other failures may result due to computer platforms that are unable to store the year as four-digits. The following sections discuss the Y2k Problem with respect to the NSV system.

5.1 Statement

The source code authored under this contract is believed by its authors to be fully year 2000 compliant. This conclusion is based on specific software design choices, on specific tests conducted and on a review of all date handling source code. The executable programs delivered under this contract are expected to behave in a fully compliant manner provided that separately purchased third-party components and libraries are themselves truly compliant.

5.2 Date/Time Usage

NSV source code has been standardized to use a single date/time format when expressed as a string. This format has a four-digit year and may be expressed by one of the following specifications, depending upon the context.

1) `yyyy-MM-dd HH:mm:ss` (used in `CGXDateTimeCtrl` custom formats)
2) `%Y-%m-%d %H:%M:%S` (used in `COleDateTime.Format()` calls)
3) `%4.4d-%2.2d-%2.2d %2.2d:%2.2d:%2.2d` (used in `sprintf()` calls)
4) `YYYY-MM-DD HH24:MI:SS` (used in Oracle `TO_DATE()` SQL)

5.2.1 Java Server

The method `CDBSDefinitions.getTimeFormat()` provides the SQL syntax for referencing the current time within a specific DBMS.

The method `CDBSDefinitions.formatDateString()` wraps the pre-formatted date/time string with the proper DBMS-specific SQL syntax for update/insert of date/time fields. This method may or may not use the format specification #4 above, depending on the DBMS type.

`CDataSet` has a string data member timestamp (in standardized format) holding the time of the last query. Before any query, the timestamp is initialized with a hard-coded string in the standardized format. Then, to retrieve newly updated or inserted records, the timestamp is used to perform an SQL date comparison within DBMS using the '>' SQL operator.

Trace messages are stamped with the current time using `System.currentTimeMillis()` and `java.util.Date`. The `toString()` method of the latter class does not meet the NSV standardized format, but the resulting string is merely used in trace/log print statements and is not used by any program data or logic.

`ResultSet.getString()` is used to get the formatted date/time string from a DBMS field. The NSV standardized format was chosen to match the format returned here.

`System.currentTimeMillis()` is used for timestamping within the `CObjectPool` class. The class does a subtraction of one timestamp from another to determine the expiration of a timeout period. It also compares the size of timeout periods with '>' operator.

5.2.2 C++ Client

"`Core`" `DataSet`s pass date/time data fields during client-server communication as strings, without performing any formatting, interpretation, calculating, comparing or sequencing.

The method `CCriteriaProductStruct::makeCriteria()` accepts a `time_t` parameter and converts it to a `struct tm` using the standard C library `localtime()` function. It then uses format specification #3 above, adding 1900 to the year field as required (which is accepted Y2K practice). Note, however, that despite the presence of this method, NSV does not currently call it.

The method `CClientSession::connect()` uses the standard C library `time()` function as an arbitrary seed into a random number generator for creating random authentication challenge text.

5.2.2.1 Console

Dates are displayed as strings, exactly as received from the Server RDBMS, without any formatting, interpretation, calculating, comparing or sequencing, in the following classes.

```
AIDEEventDlg, AIDEHostDlg, AIDESessionDlg,
HostDlg, EventDlg, EventTypeDlg, EventAuditDlg
```

The only place where a date/time can be entered by a user is in the `CCriteriaGrid` class. The method `CCriteriaGrid::OnValidateCell()` creates the Stringray `CGXDateTimeCtrl` date/time controls using format specification #1 above. It then uses `COleDateTime::GetCurrentTime()` and `COleDateTime::Format()` (without parameters) to initialize the Stringray control to the current date/time. The method `CCriteriaGrid::syncContents()` uses `CGXDateTimeCtrl::GetDateTime()` to get the user's data/time entry as a `COleDateTime`, and then calls `COleDateTime::Format()` using format specification #2 above to get a string in standard format. The method `CCriteriaGrid::storeToFile()` uses `CGXDateTimeCtrl::GetDateTime()` to get the user's data/time entry as a `COleDateTime`, and then calls `COleDateTime::Format()` using format specification #2 above to get a string in standard format.

The method `CCriteriaGrid::setColorsByRandom()` uses the standard C library `time()` function as an arbitrary seed into a random number generator for creating random colors.

Event callbacks from WorldView get a timestamp parameter of type `double`, but it is not used by NSV functions.

Scene object `PROTO`s have VRML exposed fields of type `SFTime`. These are used to specify animation periods, specifically, a relative interval in seconds.

5.2.2.2 Cartridge

In CartridgeAPI.cpp certain object constructors initialize a formatted date/time string using a hard-coded date in the standardized format identified above.

The `CJobs` class uses `COleDateTime` to hold the ISS job's start and end date/time. These times are acquired directly into the `COleDateTime` variables from the ISS database through a DAO record-set using the `DFX_DateTime()` function.

The `CISSTranslator` class transfers date/times as strings, converting them by calling `COleDateTime::Format()`using format specification #2 above to get a string in standard format.

5.3 Third-Party Libraries

Sun Microsystems claims that the JDK 1.2.1 is compliant. Inprise claims that Visibroker 3.4 for Java and Visibroker 3.3 for C++ are compliant. RogueWave claims that Objective Grid 6.0 and Objective Toolkit 7.0 are compliant.

5.3.1 Java Details

`java.util.Date` is a class intended to reflect coordinated universal time (UTC), although it may not do so exactly depending on the host environment of the Java Virtual Machine. Most formatting methods of this class have been deprecated in favor of the `Calendar` and/or `DateFormat` classes. It converts via `toString()` to the following format "`dow mon dd hh:mm:ss zzz yyyy`".

`System.currentTimeMillis()` returns a `long` holding the difference (in milliseconds) between the current time and midnight, January 1, 1970 UTC.

5.3.2 C++ Details

`time_t` is a standard C library type defined to hold seconds since an epoch. Based on its size and the definition of the epoch, it is defined to hold date/times within the range January 1, 00:00:00, 1970 to January 18, 19:14:07, 2038.

`localtime()` is a standard C library function that converts a `time_t` to a `struct tm`. This structure has separate fields for each component of a date/time and fully supports the date/time range of `time_t`. Note that the year field contains the actual year minus 1900, meaning that the year 2000 is held as the value 100. Also note that C standard library functions are capable of accounting for Daylight Savings Time.

A `COleDateTime` object encapsulates the OLE automation `DATE` type, which is implemented as a floating-point value measuring days from midnight, 30 December 1899. So, the `COleDateTime` class handles dates from 1 January 100 to 31 December 9999, with a time resolution of roughly 1 millisecond. However, `COleDateTime` ignores Daylight Saving Time.

`CGXDateTimeControl` is a Stingray Objective Grid control that is based on `COleDateTime`. The graphical user entry supports the year 2000 and recognizes it as a leap year.

`DFX_DateTime()` is a function that marshalls a `COleDateTime` from a field in a `CDaoRecordSet`. `CDaoRecordSet` accesses the database through ODBC and an ODBC driver for a given DBMS (MS Access97, in the case of ISS).

5.4 Testing

The NSV system has been designed to handle date information correctly by always storing, manipulating, and representing the year with four-digits. Certain components of the NSV system were built with third party software. Since the source code for this software is not under our direct control, Y2K testing considers the behavior of these components in addition to NSV own source code.

5.4.1 Plan

The design and performance of various tests were aimed at verifying the behavior of the NSV system as a whole with respect to the Y2K problem. To perform these tests, the NSV database was populated with data containing dates before and after the year 2000, and NSV was run with the computer system clock set before and after the year 2000. The executed tests and their results are summarized below.

1. Scenes were successfully built from data made up of both pre-2000 and post-2000 dates. The contents of these scenes were verified against the data stored in the database.
2. Scenes were successfully built from data made up of both pre-2000 and post-2000 dates while restricting the scene content with DataSet criteria that utilized date comparison operators (such as <, >, >=). The contents of these scenes were verified against the data stored in the database.
3. The system's ability to recognize the year 2000 as a leap year was successfully tested by specifying February 29th, 2000 for post-2000 date comparisons in the above test (2).
4. The system's ability to manipulate post-2000 data was successfully tested by modifying the dates associated with scene content via the Console. The new post-2000 dates were both represented and stored correctly by the NSV system.
5. The preceding tests were performed successfully by executing both the NSV Server and Console in the year 1999, and then again in the year 2000. This was accomplished by setting the system clock ahead to March 1, 2000.
6. The preceding tests were performed successfully by executing the Console in the year 1999 and executing the NSV Server in the year 2000.

Since the NSV Server may be configured for an AIDE database, that uses Oracle, as well as the native NSV database, that uses Access, an analogous set of tests was successfully performed using an NSV Server configured for an AIDE database.

5.4.2 Summary of Results

All testing was performed on computers that are considered to be Y2k complaint. Since no error situations resulted from the battery of Y2K tests listed above, we find no evidence that the NSV system will fail due to Y2K issues. However, the system may not perform as expected if any of its components are installed on computers that are not considered to be Y2K complaint.

6 Summation

The purpose of this Phase II SBIR was to develop a functional prototype employing 3-D visualization concepts to enhance the task of network security monitoring. The attainment of the specified objectives resulted in a usable system that demonstrates the proposed capability and allows for a practical evaluation of its benefits.

All of the key aspects of the network security visualization concept were implemented and demonstrated by the prototype system.

- A distributed, extensible application architecture
- Acquiring security data from a real network by integrating with a third-party “sensor”
- An effective mapping of network security data into 3-D visualization concepts
- Presenting security data in scalable, configurable 3-D scene designs
- Integration of the 3-D visualization with the AFRL AIDE/EPIC system

